Table of Contents

1. Introduction

FunRides Car Rental, established in June 2020, is a car rental company based in the United Kingdom. The fleet consists of 150 vehicles spread across 10 branches. Its business involves renting cars to customers and providing additional services such as GPS and child safety seats.

To maximise revenue by increasing the vehicle utilisation rate and making data-driven decisions regarding vehicle procurement based on revenue trends, demand for vehicles and customer preferences, a data product was developed with a well-structured database that records comprehensive information related to vehicles, customers, and rental orders.

The report intends to describe the database's design and extract significant findings that align with the data product’s development purpose. The following sections will discuss the database framework and present three key analyses derived from five distinct business reports to provide critical recommendations for operational strategies.

2. Database Design

A new database file, **fun_rides_car_rental.db**, was created by establishing a connection with Python’s **sqlite3.connect()** function. This setup provides the essential interface for executing SQL commands such as creating tables, loading data, and running queries. We identified ten entities and their attributes in the database (see Appendix A).

2.1 Entity Relationship Diagram

This diagram (Figure 1) uses Crow’s Foot notation (Dybka, 2016) and displays all 10 tables in the database with their relationships. Moreover, the primary keys (PK) and foreign keys (FK) are clearly indicated in the figure.

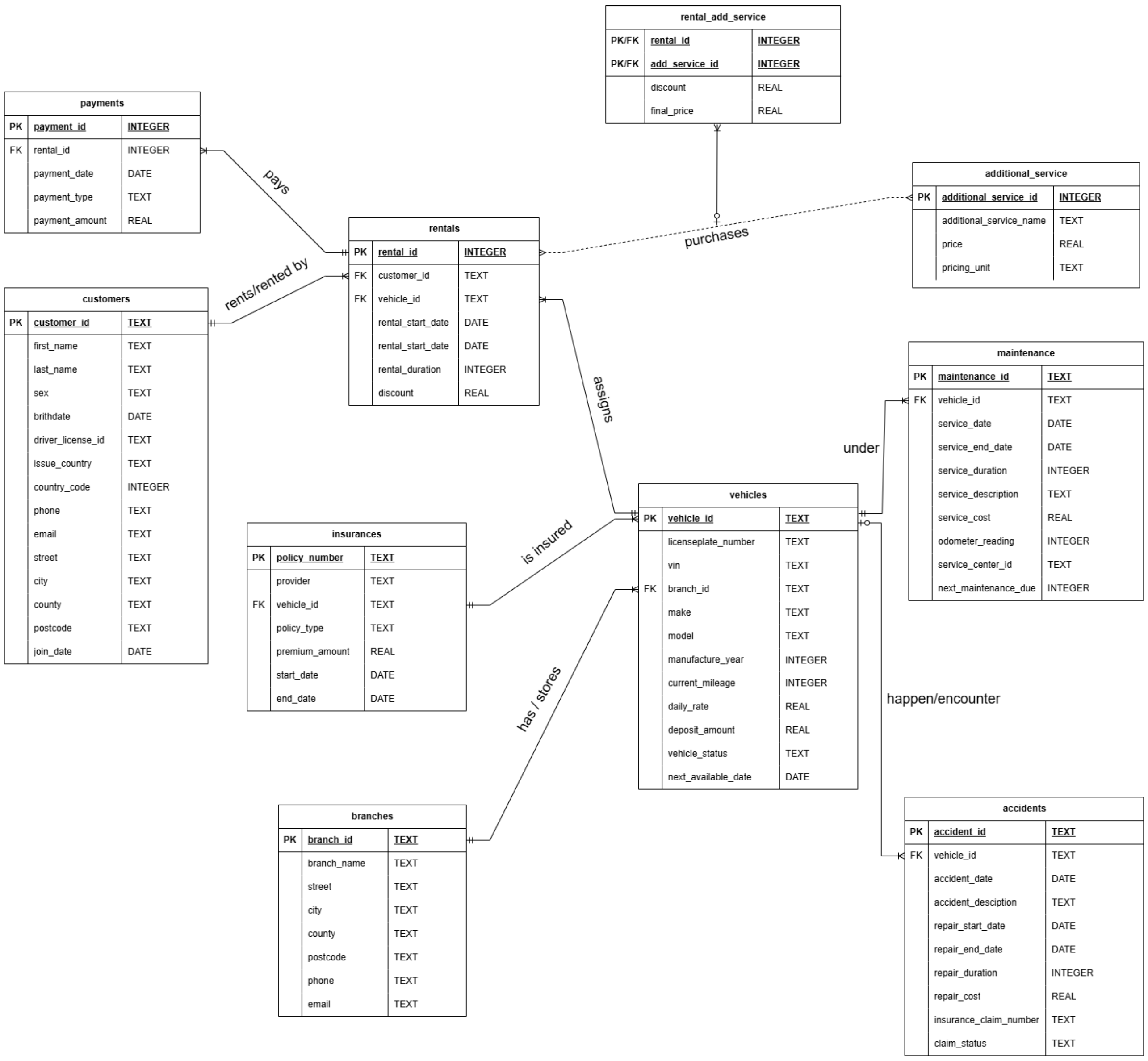


Figure 1. Entity Relationship Diagram

2.2 Relationships and Cardinalities

Entity1	Entity2	Relationships	Explanation
customers	rentals	1:N	One customer may have more than one rental order.
rentals	vehicles	N:1	Only one vehicle is assigned to each rental, and one vehicle might correspond to more than one rental.
rentals	additional_services	M:N	A rental may include more than one additional service, and each type of additional service might match more than one rental. To address this many-to-many relationship, a junction table called rental_add_service is added.
rentals	payments	1:N	For each rental, multiple payment records are possible.

vehicles	branches	N:1	Each branch contains multiple vehicles.
vehicles	insurances	1:N	Each vehicle must have basic insurance annually. Multiple insurance records may exist for vehicles purchased more than one year ago.
vehicles	maintenances	1:N	Each vehicle has undergone maintenance at least once.
vehicles	accidents	1(optional):N	Some vehicles may have experienced one or more accidents.

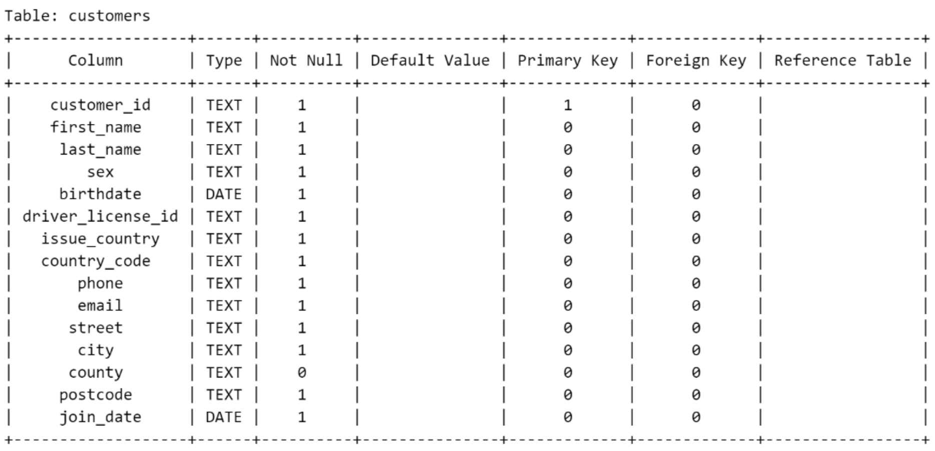
Table 1: Relationships and Cardinalities Between Two Tables

2.3 Schema Design

Using SQL Data Definition Language (DDL) in the Python environment, identified entities and their attributes discussed above are created

Customers

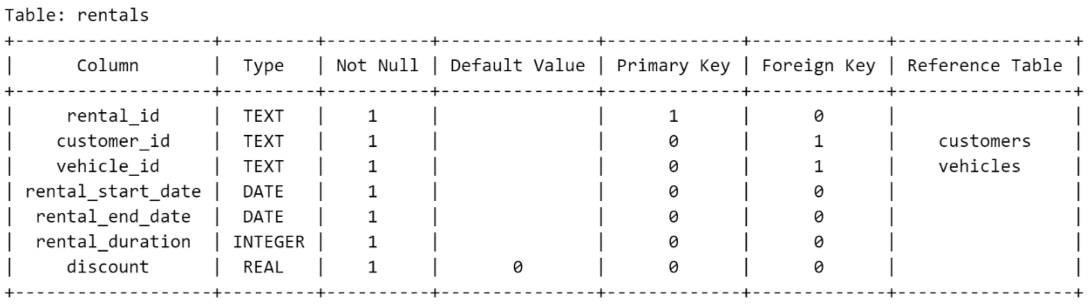
As the county of residence and email address are optional fields, we allow NULL values. Moreover, we use the CHECK function to the sex column, restricting values to male (M) or female (F) only.



```
Table: customers
+------------------+------+----------+---------------+-------------+-------------+-----------------+
|      Column      | Type | Not Null | Default Value | Primary Key | Foreign Key | Reference Table |
+------------------+------+----------+---------------+-------------+-------------+-----------------+
|    customer_id   | TEXT |    1     |               |      1      |      0      |                 |
|    first_name    | TEXT |    1     |               |      0      |      0      |                 |
|     last_name    | TEXT |    1     |               |      0      |      0      |                 |
|       sex        | TEXT |    1     |               |      0      |      0      |                 |
|     birthdate    | DATE |    1     |               |      0      |      0      |                 |
| driver_license_id| TEXT |    1     |               |      0      |      0      |                 |
|   issue_country  | TEXT |    1     |               |      0      |      0      |                 |
|   country_code   | TEXT |    1     |               |      0      |      0      |                 |
|       phone      | TEXT |    1     |               |      0      |      0      |                 |
|       email      | TEXT |    1     |               |      0      |      0      |                 |
|      street      | TEXT |    1     |               |      0      |      0      |                 |
|       city       | TEXT |    1     |               |      0      |      0      |                 |
|      county      | TEXT |    0     |               |      0      |      0      |                 |
|     postcode     | TEXT |    1     |               |      0      |      0      |                 |
|     join_date    | DATE |    1     |               |      0      |      0      |                 |
+------------------+------+----------+---------------+-------------+-------------+-----------------+
```

Figure 2. Customers data schema

Rentals

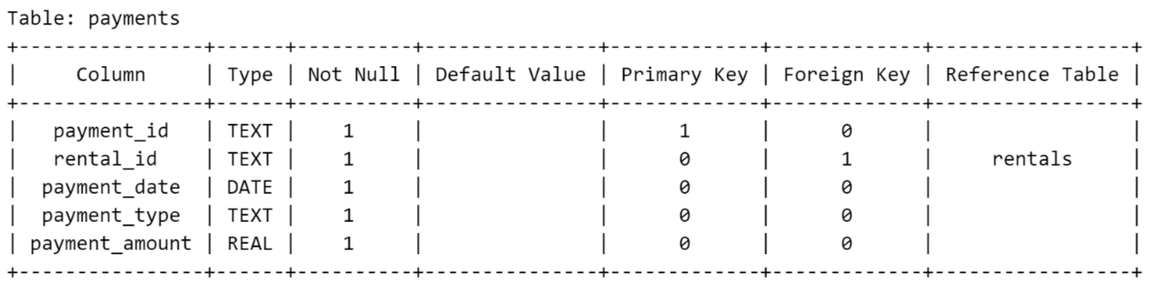


```
Table: rentals
+-------------------+---------+----------+---------------+-------------+-------------+-----------------+
|      Column       |  Type   | Not Null | Default Value | Primary Key | Foreign Key | Reference Table |
+-------------------+---------+----------+---------------+-------------+-------------+-----------------+
|     rental_id     |  TEXT   |    1     |               |      1      |      0      |                 |
|    customer_id    |  TEXT   |    1     |               |      0      |      1      |    customers    |
|     vehicle_id    |  TEXT   |    1     |               |      0      |      1      |    vehicles     |
| rental_start_date |  DATE   |    1     |               |      0      |      0      |                 |
|  rental_end_date  |  DATE   |    1     |               |      0      |      0      |                 |
|  rental_duration  | INTEGER |    1     |               |      0      |      0      |                 |
|      discount     |  REAL   |    1     |       0       |      0      |      0      |                 |
+-------------------+---------+----------+---------------+-------------+-------------+-----------------+
```

Figure 3. Rentals data schema

Payments

The payment_type column is restricted to the values: cash, credit card, debit card, and online transfer.

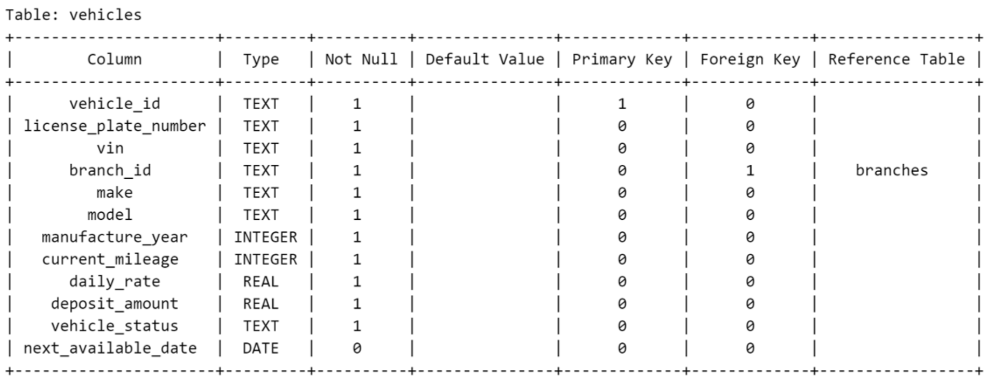


```
Table: payments
+----------------+------+----------+---------------+-------------+-------------+-----------------+
|     Column     | Type | Not Null | Default Value | Primary Key | Foreign Key | Reference Table |
+----------------+------+----------+---------------+-------------+-------------+-----------------+
|   payment_id   | TEXT |    1     |               |      1      |      0      |                 |
|   rental_id    | TEXT |    1     |               |      0      |      1      |     rentals     |
|  payment_date  | DATE |    1     |               |      0      |      0      |                 |
|  payment_type  | TEXT |    1     |               |      0      |      0      |                 |
| payment_amount | REAL |    1     |               |      0      |      0      |                 |
+----------------+------+----------+---------------+-------------+-------------+-----------------+
```

Figure 4. Payments data schema

Vehicles

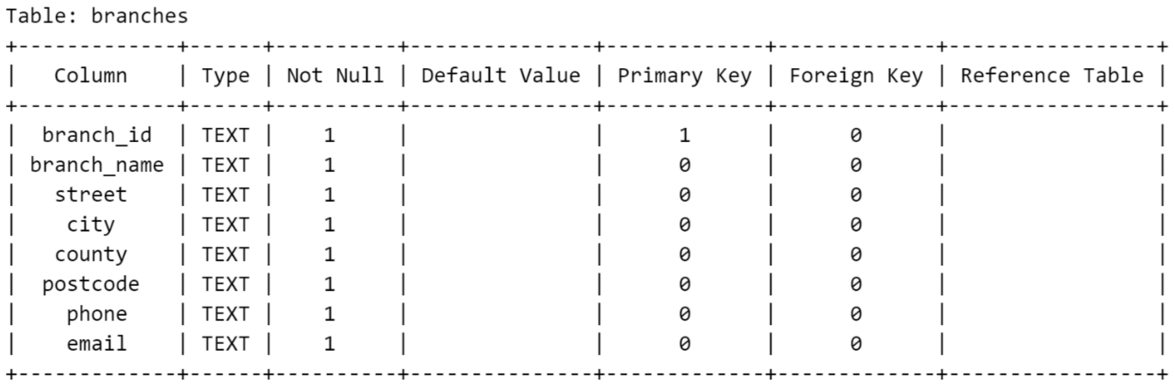
As totalled vehicles are recorded in the table, NULL values are allowed in the next_available_date column. Vehicles have available, rented, servicing, and totalled statuses, using the CHECK function to constrain.



```
Table: vehicles
+---------------------+---------+----------+---------------+-------------+-------------+-----------------+
|       Column        |  Type   | Not Null | Default Value | Primary Key | Foreign Key | Reference Table |
+---------------------+---------+----------+---------------+-------------+-------------+-----------------+
|     vehicle_id      |  TEXT   |    1     |               |      1      |      0      |                 |
| license_plate_number|  TEXT   |    1     |               |      0      |      0      |                 |
|         vin         |  TEXT   |    1     |               |      0      |      0      |                 |
|      branch_id      |  TEXT   |    1     |               |      0      |      1      |    branches     |
|        make         |  TEXT   |    1     |               |      0      |      0      |                 |
|        model        |  TEXT   |    1     |               |      0      |      0      |                 |
|   manufacture_year  | INTEGER |    1     |               |      0      |      0      |                 |
|   current_mileage   | INTEGER |    1     |               |      0      |      0      |                 |
|      daily_rate     |  REAL   |    1     |               |      0      |      0      |                 |
|    deposit_amount   |  REAL   |    1     |               |      0      |      0      |                 |
|    vehicle_status   |  TEXT   |    1     |               |      0      |      0      |                 |
| next_available_date |  DATE   |    0     |               |      0      |      0      |                 |
+---------------------+---------+----------+---------------+-------------+-------------+-----------------+
```

Figure 5. Vehicles data schema

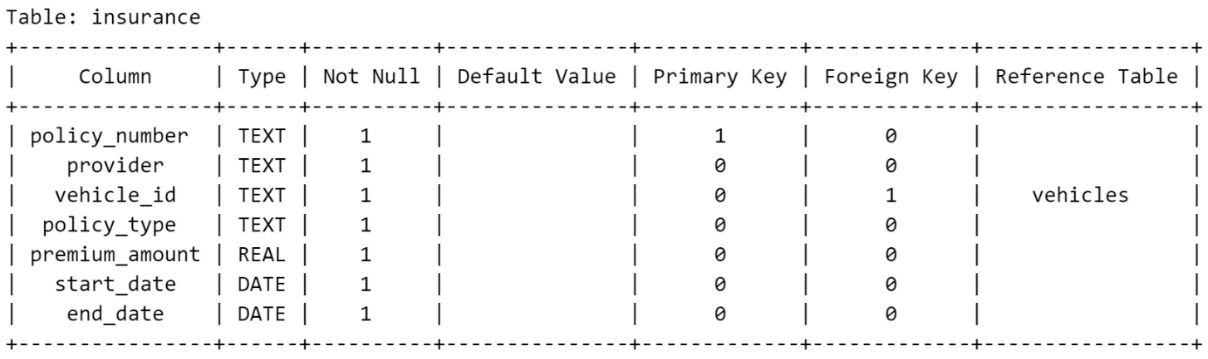
Branches



```
Table: branches
+-------------+------+----------+---------------+-------------+-------------+-----------------+
|   Column    | Type | Not Null | Default Value | Primary Key | Foreign Key | Reference Table |
+-------------+------+----------+---------------+-------------+-------------+-----------------+
|  branch_id  | TEXT |    1     |               |      1      |      0      |                 |
| branch_name | TEXT |    1     |               |      0      |      0      |                 |
|   street    | TEXT |    1     |               |      0      |      0      |                 |
|    city     | TEXT |    1     |               |      0      |      0      |                 |
|   county    | TEXT |    1     |               |      0      |      0      |                 |
|  postcode   | TEXT |    1     |               |      0      |      0      |                 |
|    phone    | TEXT |    1     |               |      0      |      0      |                 |
|    email    | TEXT |    1     |               |      0      |      0      |                 |
+-------------+------+----------+---------------+-------------+-------------+-----------------+
```

Figure 6. Branches data schema

Insurance

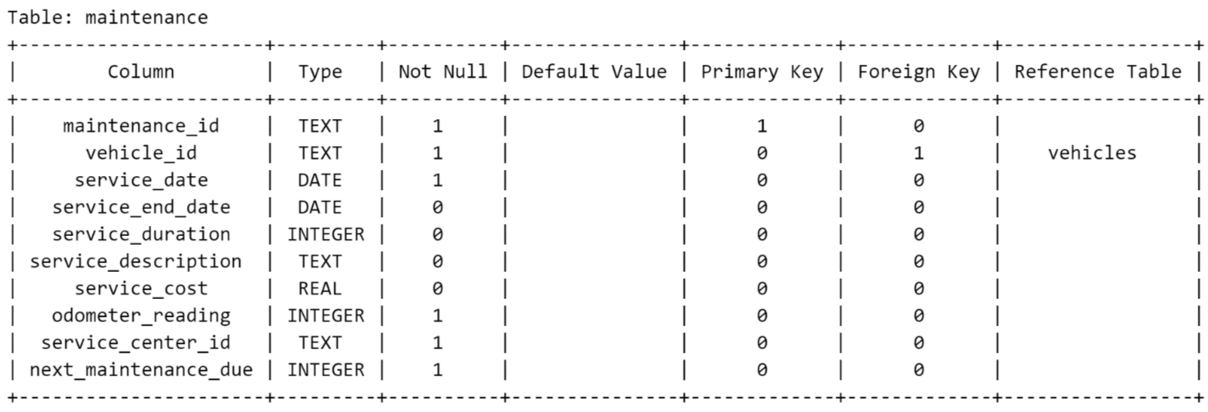


```
Table: insurance
+----------------+------+----------+---------------+-------------+-------------+-----------------+
|     Column     | Type | Not Null | Default Value | Primary Key | Foreign Key | Reference Table |
+----------------+------+----------+---------------+-------------+-------------+-----------------+
| policy_number  | TEXT |    1     |               |      1      |      0      |                 |
|    provider    | TEXT |    1     |               |      0      |      0      |                 |
|   vehicle_id   | TEXT |    1     |               |      0      |      1      |    vehicles     |
|  policy_type   | TEXT |    1     |               |      0      |      0      |                 |
| premium_amount | REAL |    1     |               |      0      |      0      |                 |
|   start_date   | DATE |    1     |               |      0      |      0      |                 |
|    end_date    | DATE |    1     |               |      0      |      0      |                 |
+----------------+------+----------+---------------+-------------+-------------+-----------------+
```

Figure 7. Insurance data schema

Maintenance

Some vehicles may have been undergoing service during the data generation period. Therefore, the service_end_date column is allowed to contain NULL values.

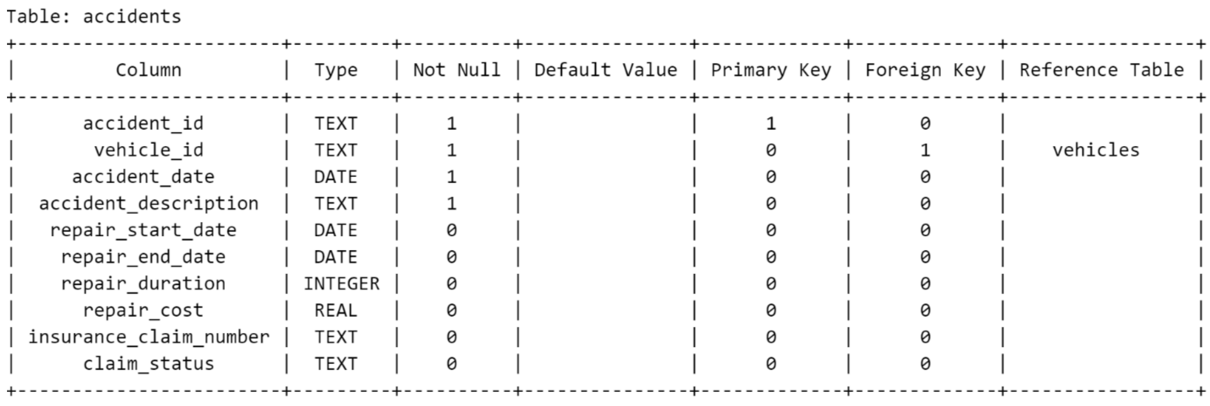


```
Table: maintenance
+---------------------+---------+----------+---------------+-------------+-------------+-----------------+
|       Column        |  Type   | Not Null | Default Value | Primary Key | Foreign Key | Reference Table |
+---------------------+---------+----------+---------------+-------------+-------------+-----------------+
|   maintenance_id    |  TEXT   |    1     |               |      1      |      0      |                 |
|     vehicle_id      |  TEXT   |    1     |               |      0      |      1      |    vehicles     |
|    service_date     |  DATE   |    1     |               |      0      |      0      |                 |
|  service_end_date   |  DATE   |    0     |               |      0      |      0      |                 |
|  service_duration   | INTEGER |    0     |               |      0      |      0      |                 |
| service_description |  TEXT   |    0     |               |      0      |      0      |                 |
|    service_cost     |  REAL   |    0     |               |      0      |      0      |                 |
|  odometer_reading   | INTEGER |    1     |               |      0      |      0      |                 |
|  service_center_id  |  TEXT   |    1     |               |      0      |      0      |                 |
| next_maintenance_due| INTEGER |    1     |               |      0      |      0      |                 |
+---------------------+---------+----------+---------------+-------------+-------------+-----------------+
```

Figure 8. Maintenance data schema

Accidents

Vehicles may be damaged or written off due to an accident, meaning that columns from repair_start_date to claim_status are possible to contain NULL values

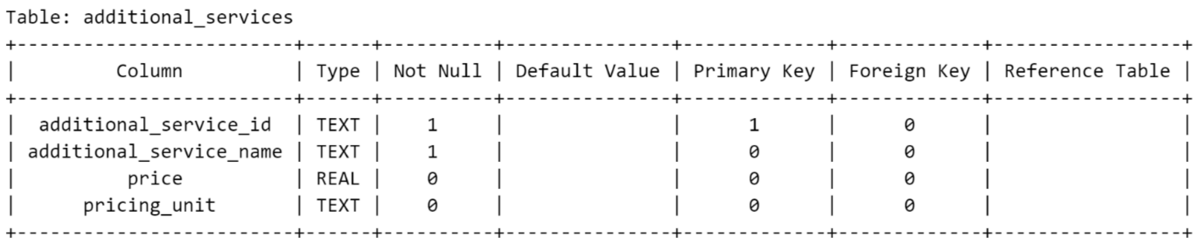


```
Table: accidents
+------------------------+---------+----------+---------------+-------------+-------------+-----------------+
|         Column         |  Type   | Not Null | Default Value | Primary Key | Foreign Key | Reference Table |
+------------------------+---------+----------+---------------+-------------+-------------+-----------------+
|      accident_id       |  TEXT   |    1     |               |      1      |      0      |                 |
|       vehicle_id       |  TEXT   |    1     |               |      0      |      1      |    vehicles     |
|     accident_date      |  DATE   |    1     |               |      0      |      0      |                 |
|  accident_description  |  TEXT   |    1     |               |      0      |      0      |                 |
|   repair_start_date    |  DATE   |    0     |               |      0      |      0      |                 |
|    repair_end_date     |  DATE   |    0     |               |      0      |      0      |                 |
|    repair_duration     | INTEGER |    0     |               |      0      |      0      |                 |
|      repair_cost       |  REAL   |    0     |               |      0      |      0      |                 |
| insurance_claim_number |  TEXT   |    0     |               |      0      |      0      |                 |
|      claim_status      |  TEXT   |    0     |               |      0      |      0      |                 |
+------------------------+---------+----------+---------------+-------------+-------------+-----------------+
```

Figure 9. Accidents data schema

Additional services

The price and pricing_unit columns are allowed to remain empty since the cost of additional insurance varies by customer.

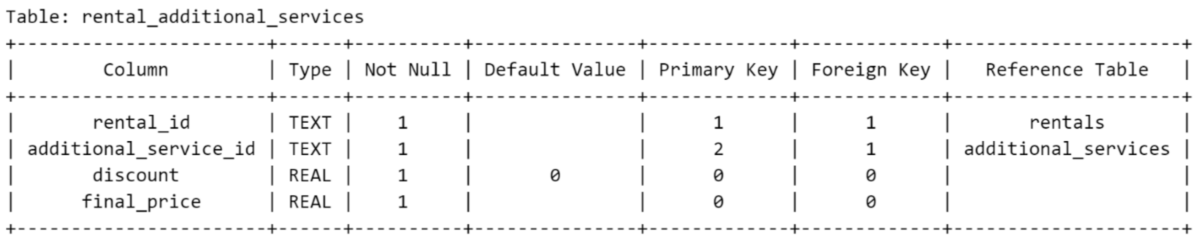


```
Table: additional_services
+------------------------+------+----------+---------------+-------------+-------------+-----------------+
|         Column         | Type | Not Null | Default Value | Primary Key | Foreign Key | Reference Table |
+------------------------+------+----------+---------------+-------------+-------------+-----------------+
|  additional_service_id | TEXT |    1     |               |      1      |      0      |                 |
| additional_service_name| TEXT |    1     |               |      0      |      0      |                 |
|         price          | REAL |    0     |               |      0      |      0      |                 |
|      pricing_unit      | TEXT |    0     |               |      0      |      0      |                 |
+------------------------+------+----------+---------------+-------------+-------------+-----------------+
```

Figure 10. Additional services data schema

Rental additional services

As a junction table, the primary key in this table is a composite key consisting of rental_id and additional_service_id.



```
Table: rental_additional_services
+----------------------+------+----------+---------------+-------------+-------------+--------------------+
|        Column        | Type | Not Null | Default Value | Primary Key | Foreign Key |  Reference Table   |
+----------------------+------+----------+---------------+-------------+-------------+--------------------+
|       rental_id      | TEXT |    1     |               |      1      |      1      |       rentals      |
| additional_service_id| TEXT |    1     |               |      2      |      1      | additional_services|
|       discount       | REAL |    1     |       0       |      0      |      0      |                    |
|      final_price     | REAL |    1     |               |      0      |      0      |                    |
+----------------------+------+----------+---------------+-------------+-------------+--------------------+
```

Figure 11. Rental additional services data schema

2.4 Normalised Datasets

The database schema has been designed to conform to the Third Normal Form (3NF). Each table is first normalised to the First Normal Form (1NF), ensuring it contains only atomic values with no repeating groups. The Second Normal Form (2NF) is achieved because every non-key attribute depends entirely on the primary key. Adhering to 3NF, the schema guarantees that every non-key attribute is directly dependent on its respective primary key, thereby eliminating any transitive dependencies (Sikora, 1997).

3. Data Preparation

Synthetic data was generated using Python to reflect operational data for the business. After loading it into an SQLite database, integrity checks confirmed that record counts, unique keys, and field values met the expected standards, ensuring a complete and consistent dataset.

3.1 Synthetic Data Generation

Python libraries such as pandas, NumPy and faker were used to generate synthetic data that reflects operational data as of March 11, 2025. The datasets were designed to accurately reflect the interconnected nature of the business operations.

Customer data assumes that 60% are tourists (The Brainy Insights, n.d.), with 30% of these lacking a UK mobile number. Age distribution is set at 15% for those aged 18 to 25, 75% for those aged 25 to 59, and 10% for those 60 and above, with joining dates scaling with fleet growth.

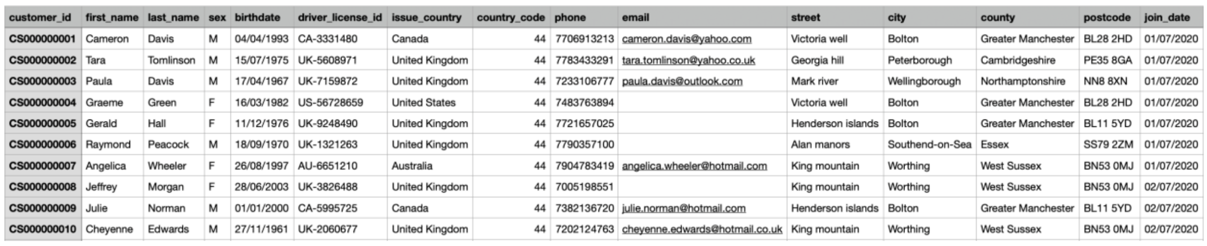
Fleet allocation was done based on real-world demand; for example, London Central has 25 vehicles while Newcastle has 10. Daily rental rates and deposit amounts vary by make, model, and manufacture year. Each vehicle is insured under a basic one-year policy from its purchase date, and customers are required to purchase additional coverage.

Over a five-year period, between 6 and 8 accidents occurred annually, and three vehicles were totalled (LNC0012 on August 12, 2023; MAN0003 on September 9, 2024; and EDI0005 on February 13, 2025), with no subsequent maintenance or rental records for these vehicles. Maintenance is scheduled every 12,000 miles, with services allowed to occur ±62 miles from the due mileage.

Load points were created in the data to reflect peak rental periods during July and August (Acko, n.d.), with discounts applied for longer rentals. Payment amounts are calculated based on rental duration, daily rates, and additional service charges, with 20% of customers using two payment methods.

Below are example rows from each dataset.

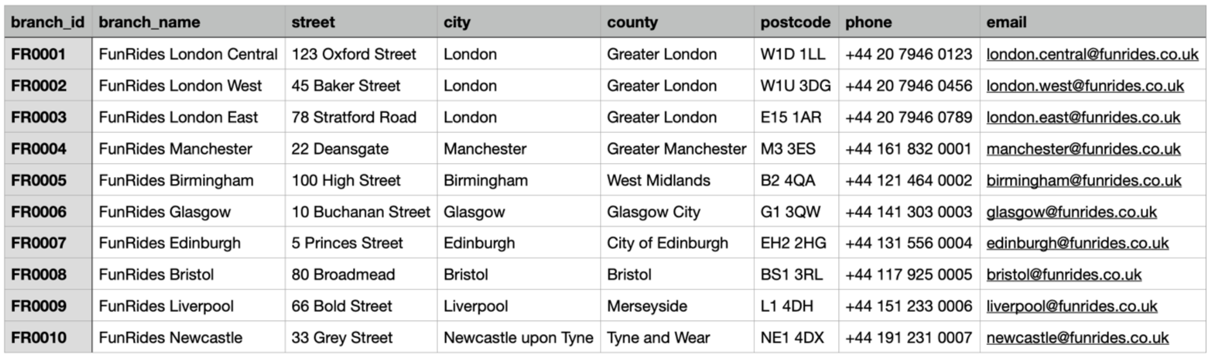
Customers



customer_id	first_name	last_name	sex	birthdate	driver_license_id	issue_country	country_code	phone	email	street	city	county	postcode	join_date
CS000000001	Cameron	Davis	M	04/04/1993	CA-3331480	Canada	44	7706913213	cameron.davis@yahoo.com	Victoria well	Bolton	Greater Manchester	BL28 2HD	01/07/2020
CS000000002	Tara	Tomlinson	M	15/07/1975	UK-5608971	United Kingdom	44	7783433291	tara.tomlinson@yahoo.co.uk	Georgia hill	Peterborough	Cambridgeshire	PE35 8GA	01/07/2020
CS000000003	Paula	Davis	M	17/04/1967	UK-7159872	United Kingdom	44	7233106777	paula.davis@outlook.com	Mark river	Wellingborough	Northamptonshire	NN8 8XN	01/07/2020
CS000000004	Graeme	Green	F	16/03/1982	US-56728659	United States	44	7483763894		Victoria well	Bolton	Greater Manchester	BL28 2HD	01/07/2020
CS000000005	Gerald	Hall	F	11/12/1976	UK-9248490	United Kingdom	44	7721657025		Henderson islands	Bolton	Greater Manchester	BL11 5YD	01/07/2020
CS000000006	Raymond	Peacock	M	18/09/1970	UK-1321263	United Kingdom	44	7790357100		Alan manors	Southend-on-Sea	Essex	SS79 2ZM	01/07/2020
CS000000007	Angelica	Wheeler	F	26/08/1997	AU-6651210	Australia	44	7904783419	angelica.wheeler@hotmail.com	King mountain	Worthing	West Sussex	BN53 0MJ	01/07/2020
CS000000008	Jeffrey	Morgan	F	28/06/2003	UK-3826488	United Kingdom	44	7005198551		King mountain	Worthing	West Sussex	BN53 0MJ	02/07/2020
CS000000009	Julie	Norman	M	01/01/2000	CA-5995725	Canada	44	7382136720	julie.norman@hotmail.com	Henderson islands	Bolton	Greater Manchester	BL11 5YD	02/07/2020
CS000000010	Cheyenne	Edwards	M	27/11/1961	UK-2060677	United Kingdom	44	7202124763	cheyenne.edwards@hotmail.co.uk	King mountain	Worthing	West Sussex	BN53 0MJ	02/07/2020

Figure 12. Customers Dataset

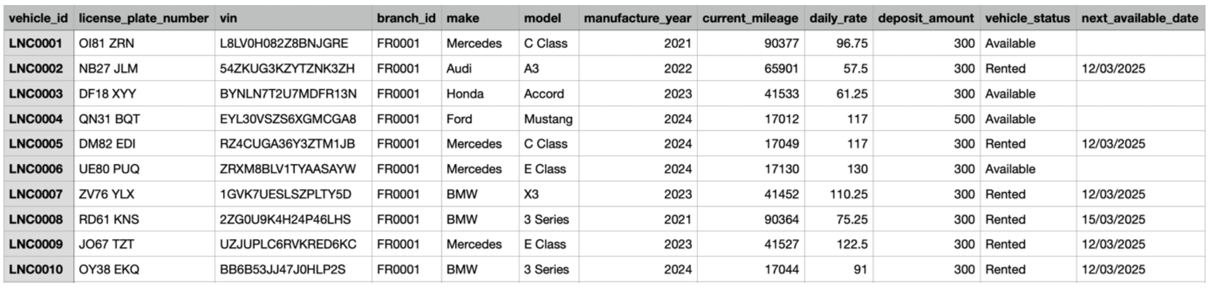
Branches



branch_id	branch_name	street	city	county	postcode	phone	email
FR0001	FunRides London Central	123 Oxford Street	London	Greater London	W1D 1LL	+44 20 7946 0123	london.central@funrides.co.uk
FR0002	FunRides London West	45 Baker Street	London	Greater London	W1U 3DG	+44 20 7946 0456	london.west@funrides.co.uk
FR0003	FunRides London East	78 Stratford Road	London	Greater London	E15 1AR	+44 20 7946 0789	london.east@funrides.co.uk
FR0004	FunRides Manchester	22 Deansgate	Manchester	Greater Manchester	M3 3ES	+44 161 832 0001	manchester@funrides.co.uk
FR0005	FunRides Birmingham	100 High Street	Birmingham	West Midlands	B2 4QA	+44 121 464 0002	birmingham@funrides.co.uk
FR0006	FunRides Glasgow	10 Buchanan Street	Glasgow	Glasgow City	G1 3QW	+44 141 303 0003	glasgow@funrides.co.uk
FR0007	FunRides Edinburgh	5 Princes Street	Edinburgh	City of Edinburgh	EH2 2HG	+44 131 556 0004	edinburgh@funrides.co.uk
FR0008	FunRides Bristol	80 Broadmead	Bristol	Bristol	BS1 3RL	+44 117 925 0005	bristol@funrides.co.uk
FR0009	FunRides Liverpool	66 Bold Street	Liverpool	Merseyside	L1 4DH	+44 151 233 0006	liverpool@funrides.co.uk
FR0010	FunRides Newcastle	33 Grey Street	Newcastle upon Tyne	Tyne and Wear	NE1 4DX	+44 191 231 0007	newcastle@funrides.co.uk

Figure 13. Branches Dataset

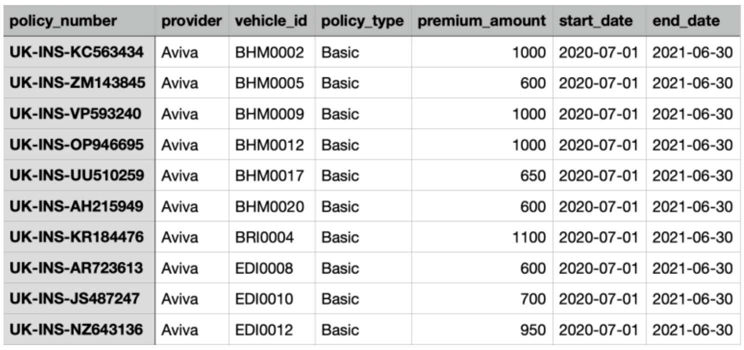
Vehicles



vehicle_id	license_plate_number	vin	branch_id	make	model	manufacture_year	current_mileage	daily_rate	deposit_amount	vehicle_status	next_available_date
LNC0001	OI81 ZRN	L8LV0H082Z8BNJGRE	FR0001	Mercedes	C Class	2021	90377	96.75	300	Available	
LNC0002	NB27 JLM	54ZKUG3KZYTZNK3ZH	FR0001	Audi	A3	2022	65901	57.5	300	Rented	12/03/2025
LNC0003	DF18 XYY	BYNLN7T2U7MDFR13N	FR0001	Honda	Accord	2023	41533	61.25	300	Available	
LNC0004	QN31 BQT	EYL30VSZS6XGMCGA8	FR0001	Ford	Mustang	2024	17012	117	500	Available	
LNC0005	DM82 EDI	RZ4CUGA36Y3ZTM1JB	FR0001	Mercedes	C Class	2024	17049	117	300	Rented	12/03/2025
LNC0006	UE80 PUQ	ZRXM8BLV1TYAASAYW	FR0001	Mercedes	E Class	2024	17130	130	300	Available	
LNC0007	ZV76 YLX	1GVK7UESLSZPLTY5D	FR0001	BMW	X3	2023	41452	110.25	300	Rented	12/03/2025
LNC0008	RD61 KNS	2ZG0U9K4H24P46LHS	FR0001	BMW	3 Series	2021	90364	75.25	300	Rented	15/03/2025
LNC0009	JO67 TZT	UZJUPLC6RVKRED6KC	FR0001	Mercedes	E Class	2023	41527	122.5	300	Rented	12/03/2025
LNC0010	OY38 EKQ	BB6B53JJ47J0HLP2S	FR0001	BMW	3 Series	2024	17044	91	300	Rented	12/03/2025

Figure 14. Vehicles Dataset

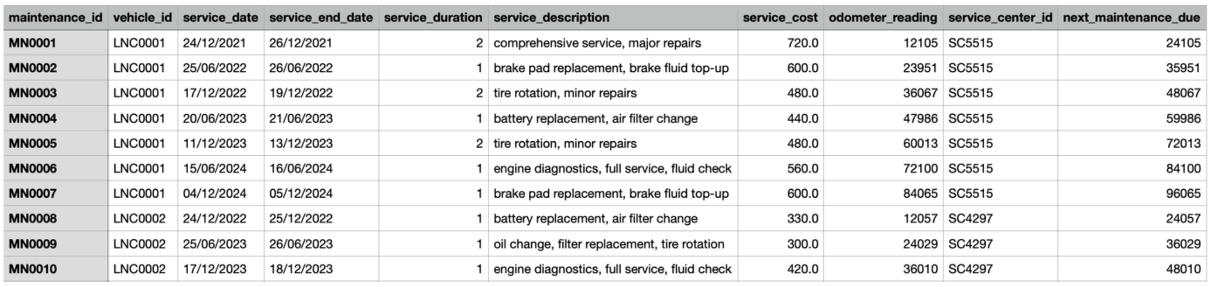
Insurance



policy_number	provider	vehicle_id	policy_type	premium_amount	start_date	end_date
UK-INS-KC563434	Aviva	BHM0002	Basic	1000	2020-07-01	2021-06-30
UK-INS-ZM143845	Aviva	BHM0005	Basic	600	2020-07-01	2021-06-30
UK-INS-VP593240	Aviva	BHM0009	Basic	1000	2020-07-01	2021-06-30
UK-INS-OP946695	Aviva	BHM0012	Basic	1000	2020-07-01	2021-06-30
UK-INS-UU510259	Aviva	BHM0017	Basic	650	2020-07-01	2021-06-30
UK-INS-AH215949	Aviva	BHM0020	Basic	600	2020-07-01	2021-06-30
UK-INS-KR184476	Aviva	BRI0004	Basic	1100	2020-07-01	2021-06-30
UK-INS-AR723613	Aviva	EDI0008	Basic	600	2020-07-01	2021-06-30
UK-INS-JS487247	Aviva	EDI0010	Basic	700	2020-07-01	2021-06-30
UK-INS-NZ643136	Aviva	EDI0012	Basic	950	2020-07-01	2021-06-30

Figure 15. Insurance Dataset

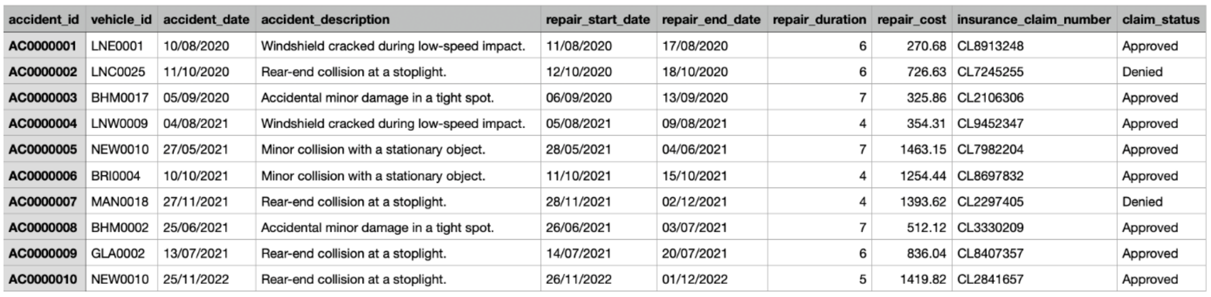
Maintenance



maintenance_id	vehicle_id	service_date	service_end_date	service_duration	service_description	service_cost	odometer_reading	service_center_id	next_maintenance_due
MN0001	LNC0001	24/12/2021	26/12/2021	2	comprehensive service, major repairs	720.0	12105	SC5515	24105
MN0002	LNC0001	25/06/2022	26/06/2022	1	brake pad replacement, brake fluid top-up	600.0	23951	SC5515	35951
MN0003	LNC0001	17/12/2022	19/12/2022	2	tire rotation, minor repairs	480.0	36067	SC5515	48067
MN0004	LNC0001	20/06/2023	21/06/2023	1	battery replacement, air filter change	440.0	47986	SC5515	59986
MN0005	LNC0001	11/12/2023	13/12/2023	2	tire rotation, minor repairs	480.0	60013	SC5515	72013
MN0006	LNC0001	15/06/2024	16/06/2024	1	engine diagnostics, full service, fluid check	560.0	72100	SC5515	84100
MN0007	LNC0001	04/12/2024	05/12/2024	1	brake pad replacement, brake fluid top-up	600.0	84065	SC5515	96065
MN0008	LNC0002	24/12/2022	25/12/2022	1	battery replacement, air filter change	330.0	12057	SC4297	24057
MN0009	LNC0002	25/06/2023	26/06/2023	1	oil change, filter replacement, tire rotation	300.0	24029	SC4297	36029
MN0010	LNC0002	17/12/2023	18/12/2023	1	engine diagnostics, full service, fluid check	420.0	36010	SC4297	48010

Figure 16. Maintenance Dataset

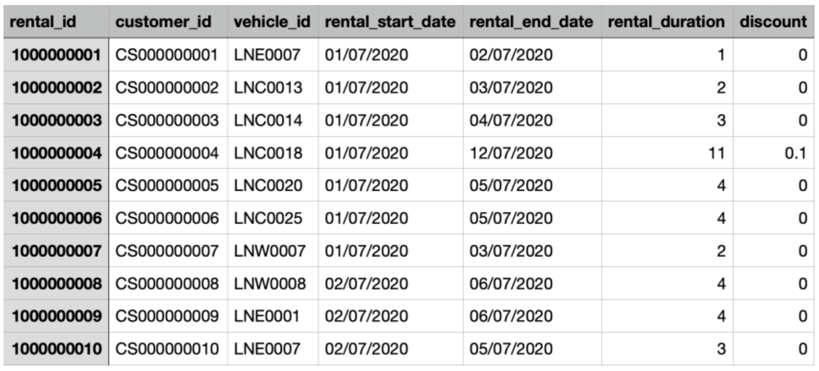
Accidents



accident_id	vehicle_id	accident_date	accident_description	repair_start_date	repair_end_date	repair_duration	repair_cost	insurance_claim_number	claim_status
AC0000001	LNE0001	10/08/2020	Windshield cracked during low-speed impact.	11/08/2020	17/08/2020	6	270.68	CL8913248	Approved
AC0000002	LNC0025	11/10/2020	Rear-end collision at a stoplight.	12/10/2020	18/10/2020	6	726.63	CL7245255	Denied
AC0000003	BHM0017	05/09/2020	Accidental minor damage in a tight spot.	06/09/2020	13/09/2020	7	325.86	CL2106306	Approved
AC0000004	LNW0009	04/08/2021	Windshield cracked during low-speed impact.	05/08/2021	09/08/2021	4	354.31	CL9452347	Approved
AC0000005	NEW0010	27/05/2021	Minor collision with a stationary object.	28/05/2021	04/06/2021	7	1463.15	CL7982204	Approved
AC0000006	BRI0004	10/10/2021	Minor collision with a stationary object.	11/10/2021	15/10/2021	4	1254.44	CL8697832	Approved
AC0000007	MAN0018	27/11/2021	Rear-end collision at a stoplight.	28/11/2021	02/12/2021	4	1393.62	CL2297405	Denied
AC0000008	BHM0002	25/06/2021	Accidental minor damage in a tight spot.	26/06/2021	03/07/2021	7	512.12	CL3330209	Approved
AC0000009	GLA0002	13/07/2021	Rear-end collision at a stoplight.	14/07/2021	20/07/2021	6	836.04	CL8407357	Approved
AC0000010	NEW0010	25/11/2022	Rear-end collision at a stoplight.	26/11/2022	01/12/2022	5	1419.82	CL2841657	Approved

Figure 17. Accidents Dataset

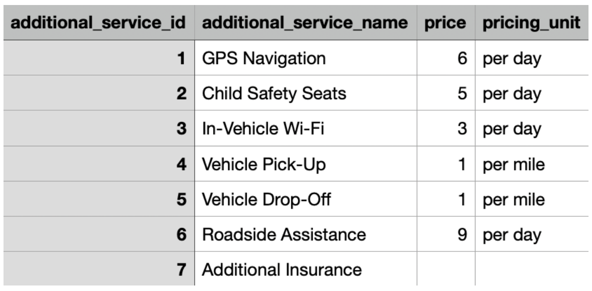
Rentals



rental_id	customer_id	vehicle_id	rental_start_date	rental_end_date	rental_duration	discount
1000000001	CS000000001	LNE0007	01/07/2020	02/07/2020	1	0
1000000002	CS000000002	LNC0013	01/07/2020	03/07/2020	2	0
1000000003	CS000000003	LNC0014	01/07/2020	04/07/2020	3	0
1000000004	CS000000004	LNC0018	01/07/2020	12/07/2020	11	0.1
1000000005	CS000000005	LNC0020	01/07/2020	05/07/2020	4	0
1000000006	CS000000006	LNC0025	01/07/2020	05/07/2020	4	0
1000000007	CS000000007	LNW0007	01/07/2020	03/07/2020	2	0
1000000008	CS000000008	LNW0008	02/07/2020	06/07/2020	4	0
1000000009	CS000000009	LNE0001	02/07/2020	06/07/2020	4	0
1000000010	CS000000010	LNE0007	02/07/2020	05/07/2020	3	0

Figure 18. Rentals Dataset

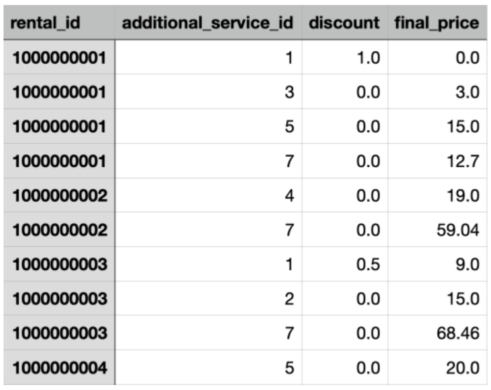
Additional Services



additional_service_id	additional_service_name	price	pricing_unit
1	GPS Navigation	6	per day
2	Child Safety Seats	5	per day
3	In-Vehicle Wi-Fi	3	per day
4	Vehicle Pick-Up	1	per mile
5	Vehicle Drop-Off	1	per mile
6	Roadside Assistance	9	per day
7	Additional Insurance		

Figure 19. Additional Services Dataset

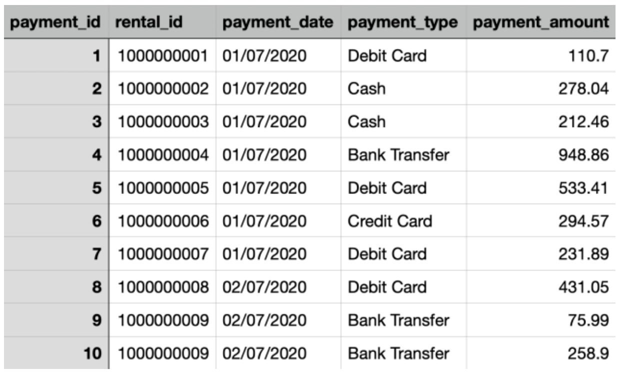
Rental Additional Services



rental_id	additional_service_id	discount	final_price
1000000001	1	1.0	0.0
1000000001	3	0.0	3.0
1000000001	5	0.0	15.0
1000000001	7	0.0	12.7
1000000002	4	0.0	19.0
1000000002	7	0.0	59.04
1000000003	1	0.5	9.0
1000000003	2	0.0	15.0
1000000003	7	0.0	68.46
1000000004	5	0.0	20.0

Figure 20. Rental Additional Services Dataset

Payments



payment_id	rental_id	payment_date	payment_type	payment_amount
1	1000000001	01/07/2020	Debit Card	110.7
2	1000000002	01/07/2020	Cash	278.04
3	1000000003	01/07/2020	Cash	212.46
4	1000000004	01/07/2020	Bank Transfer	948.86
5	1000000005	01/07/2020	Debit Card	533.41
6	1000000006	01/07/2020	Credit Card	294.57
7	1000000007	01/07/2020	Debit Card	231.89
8	1000000008	02/07/2020	Debit Card	431.05
9	1000000009	02/07/2020	Bank Transfer	75.99
10	1000000009	02/07/2020	Bank Transfer	258.9

Figure 21. Payments Dataset

3.2 Data Integrity Check

Once the data was loaded into the database, several checks were performed to ensure its quality. A record count check made sure that each table had the same number of records as the original CSV files, confirming that all data was imported correctly. A unique primary key check ensured that each table’s key field contained no duplicates, so every record is distinct.

Specific fields, such as ‘sex’, ‘payment_method’, and ‘vehicle_status’, were checked to make sure they only contained allowed values. This prevented any invalid data from entering these columns. In addition, the rental records were reviewed to verify that no two rentals overlapped for the same vehicle or customer. Overall, these checks confirm that the data is complete, consistent, and meets the business rules.

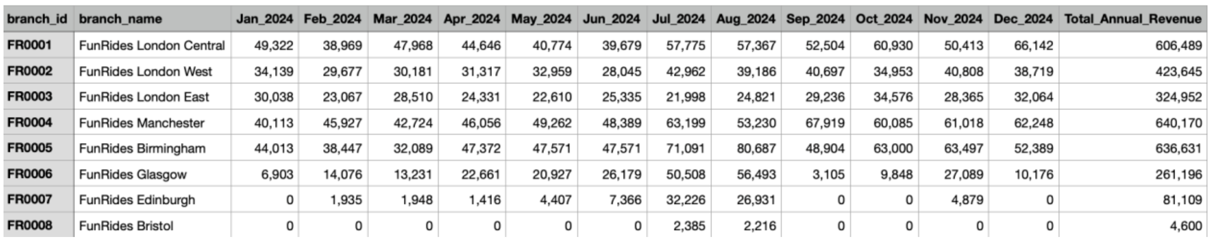
4. Business Reports

Five business reports were extracted using SQL queries to support three distinct analyses: Financial Analysis, Vehicle Utilisation, and Demand Analysis. These analyses provide strategic insights aligned with our business objectives and inform critical operational recommendations.

The Edinburgh and Bristol branches show zero or minimal rentals for several months, which could indicate data omissions, temporary closures, or low operational activity. Liverpool and Newcastle recorded no rentals in 2024, suggesting these branches may not have been operational. Consequently, their insights are excluded from this report. A further review is recommended to address these anomalies and ensure accurate analysis.

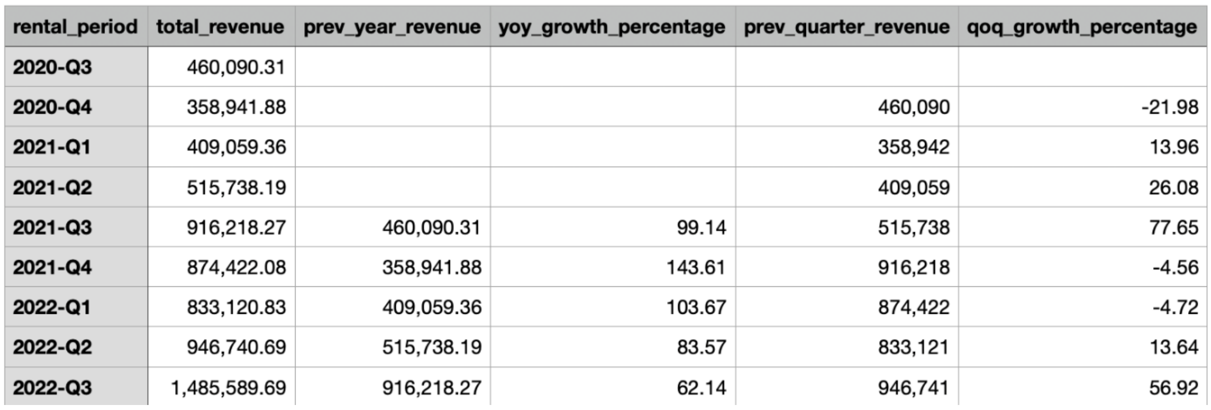
4.1 Financial Analysis

The SQL query generates the 2024 monthly revenue report by extracting the month and summing the payment amounts for each branch. The results are then grouped and ordered by branch, generating a detailed monthly revenue breakdown for the business. Additionally, a revenue report comparing the quarter-over-quarter performance was generated for the five-year period.



branch_id	branch_name	Jan_2024	Feb_2024	Mar_2024	Apr_2024	May_2024	Jun_2024	Jul_2024	Aug_2024	Sep_2024	Oct_2024	Nov_2024	Dec_2024	Total_Annual_Revenue
FR0001	FunRides London Central	49,322	38,969	47,968	44,646	40,774	39,679	57,775	57,367	52,504	60,930	50,413	66,142	606,489
FR0002	FunRides London West	34,139	29,677	30,181	31,317	32,959	28,045	42,962	39,186	40,697	34,953	40,808	38,719	423,645
FR0003	FunRides London East	30,038	23,067	28,510	24,331	22,610	25,335	21,998	24,821	29,236	34,576	28,365	32,064	324,952
FR0004	FunRides Manchester	40,113	45,927	42,724	46,056	49,262	48,389	63,199	53,230	67,919	60,085	61,018	62,248	640,170
FR0005	FunRides Birmingham	44,013	38,447	32,089	47,372	47,571	47,571	71,091	80,687	48,904	63,000	63,497	52,389	636,631
FR0006	FunRides Glasgow	6,903	14,076	13,231	22,661	20,927	26,179	50,508	56,493	3,105	9,848	27,089	10,176	261,196
FR0007	FunRides Edinburgh	0	1,935	1,948	1,416	4,407	7,366	32,226	26,931	0	0	4,879	0	81,109
FR0008	FunRides Bristol	0	0	0	0	0	0	2,385	2,216	0	0	0	0	4,600

Figure 22. Branch-Wise Revenue Report 2024



rental_period	total_revenue	prev_year_revenue	yoy_growth_percentage	prev_quarter_revenue	qoq_growth_percentage
2020-Q3	460,090.31				
2020-Q4	358,941.88			460,090	-21.98
2021-Q1	409,059.36			358,942	13.96
2021-Q2	515,738.19			409,059	26.08
2021-Q3	916,218.27	460,090.31	99.14	515,738	77.65
2021-Q4	874,422.08	358,941.88	143.61	916,218	-4.56
2022-Q1	833,120.83	409,059.36	103.67	874,422	-4.72
2022-Q2	946,740.69	515,738.19	83.57	833,121	13.64
2022-Q3	1,485,589.69	916,218.27	62.14	946,741	56.92

Figure 23. QoQ Revenue Growth Report

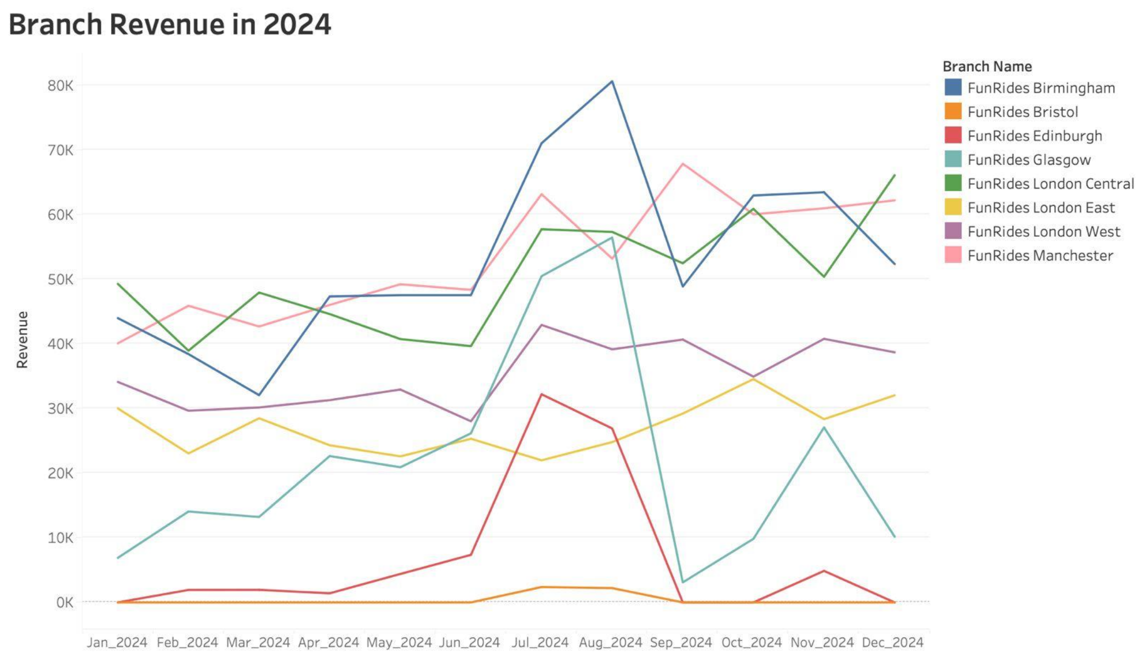


Figure 24. Branch Revenue Report 2024

Figure 24 shows that the Birmingham and Manchester branches were the top performers, generating annual revenues of £636,630.58 and £640,170.41, respectively. The London Central branch followed with £606,488.70 in revenue. The Glasgow branch, with £261,196.39 in annual revenue, experienced a huge revenue decline from August to September.

The report also reveals strong seasonality, with nearly every location reaching peak revenue in July and August. For example, FunRides London Central's revenue jumps from £39,679 in June to £57,775 in July and remains elevated at £57,367 in August. These peaks reflect increased tourist demand during the summer months.

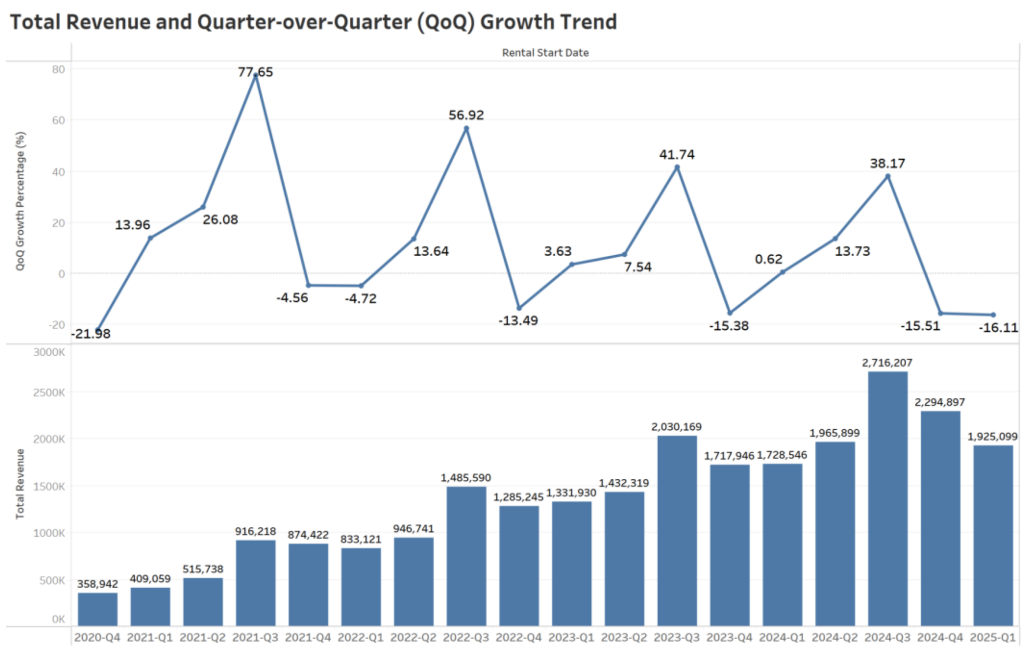


Figure 25. QoQ Growth Trend

Figure 25 shows a decline in revenue growth over the years and highlights pronounced seasonality.

4.2 Vehicle Utilisation Analysis

The SQL query calculates vehicle utilisation rates for each branch in 2024 by dividing total rental days by available days, factoring in maintenance and accident durations. It joins data from multiple tables (vehicles, branches, rentals, maintenance, accidents) and computes average utilisation per branch, ordered by branch ID. For rentals starting in 2024 but ending in 2025, the query caps the rental duration on 31st December 2024.

$$Utilisation \ = \frac{Number\ of\ days\ car\ was\ rented}{366 - (Servicing\ days\ + Repair\ days)}$$

branch_id	branch_name	vehicle_utilisation
FR0001	FunRides London Central	99%
FR0002	FunRides London West	99%
FR0003	FunRides London East	97%
FR0004	FunRides Manchester	94%
FR0005	FunRides Birmingham	82%
FR0006	FunRides Glasgow	47%

Figure 26. Vehicle Utilisation Report

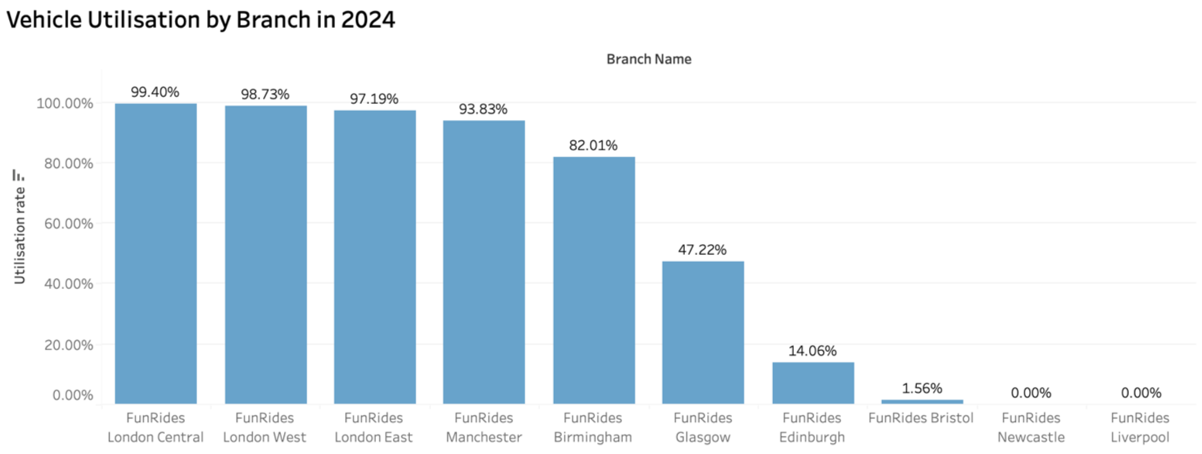


Figure 27. Vehicle Utilisation by Branch

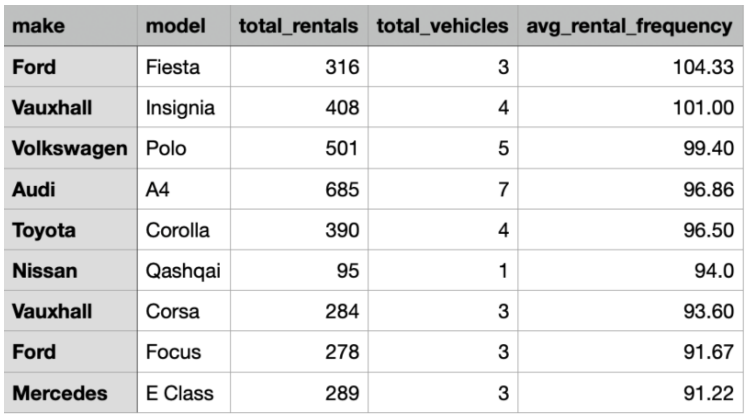
As observed in Figure 27, branches in London Central (FR0001), London West (FR0002), and London East (FR0003) demonstrate utilisation rates exceeding 97%, reflecting high demand and effective fleet management.

In contrast, branches such as Manchester (FR0004), Birmingham (FR0005), and Glasgow (FR0006) show utilisation rates ranging from 47% to 82%, performing decently but with potential for further optimisation.

4.3 Demand Analysis

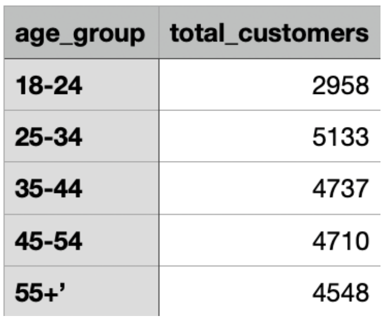
Three business reports were generated. The first report features an SQL query that calculates the average number of rentals in 2024 by grouping the data by make and model, achieved through a join between the vehicles and rentals tables. The other two reports focus on customer segmentation, employing SQL queries that determine age ranges based on customers' dates of birth and gender.

$$Average\ Rental\ Frequency = \frac{No.\,of\ rentals\ for\ a\ model}{Total\ no.\,of\ that\ model\ in\ the\ fleet}$$



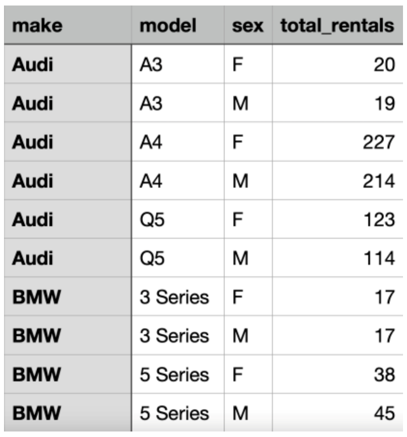
make	model	total_rentals	total_vehicles	avg_rental_frequency
Ford	Fiesta	316	3	104.33
Vauxhall	Insignia	408	4	101.00
Volkswagen	Polo	501	5	99.40
Audi	A4	685	7	96.86
Toyota	Corolla	390	4	96.50
Nissan	Qashqai	95	1	94.0
Vauxhall	Corsa	284	3	93.60
Ford	Focus	278	3	91.67
Mercedes	E Class	289	3	91.22

Figure 28. Vehicle Rental Averages Report



age_group	total_customers
18-24	2958
25-34	5133
35-44	4737
45-54	4710
55+'	4548

Figure 29. Customer Age Report



make	model	sex	total_rentals
Audi	A3	F	20
Audi	A3	M	19
Audi	A4	F	227
Audi	A4	M	214
Audi	Q5	F	123
Audi	Q5	M	114
BMW	3 Series	F	17
BMW	3 Series	M	17
BMW	5 Series	F	38
BMW	5 Series	M	45

Figure 30. Gender-Based Vehicle Demand Report

As shown in Figure 31, both the Ford Fiesta and the Vauxhall Insignia achieve an average of over 100 rentals per vehicle, highlighting their popularity and advantages.

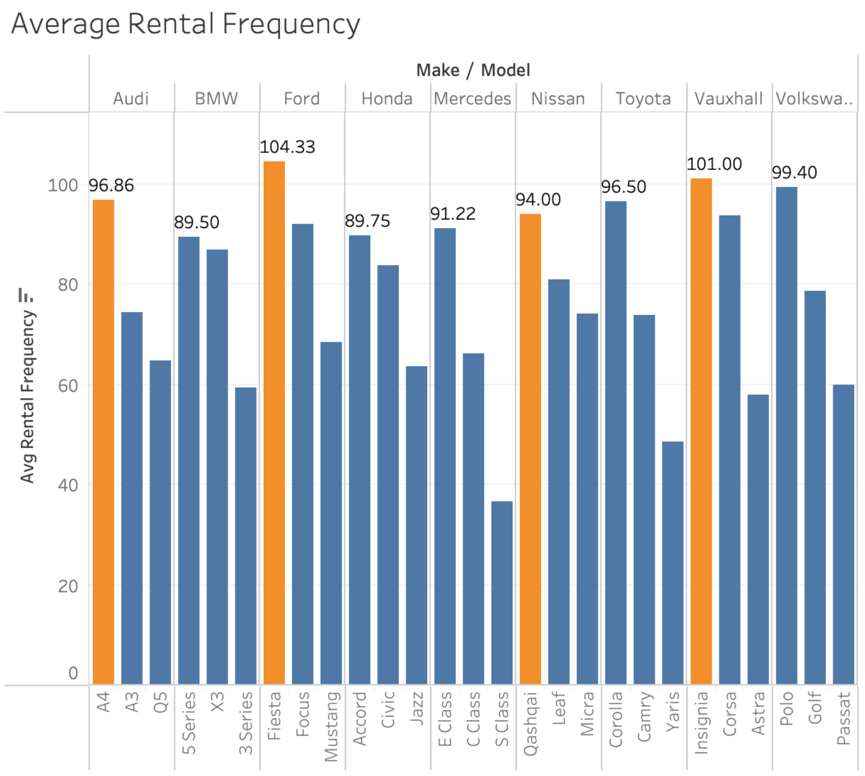


Figure 31. Average Rental Frequency

As shown in Figure 32, the primary consumer base consists of drivers aged 25–54 (66%), a demographic facing financial pressures related to housing, family, and career advancement.

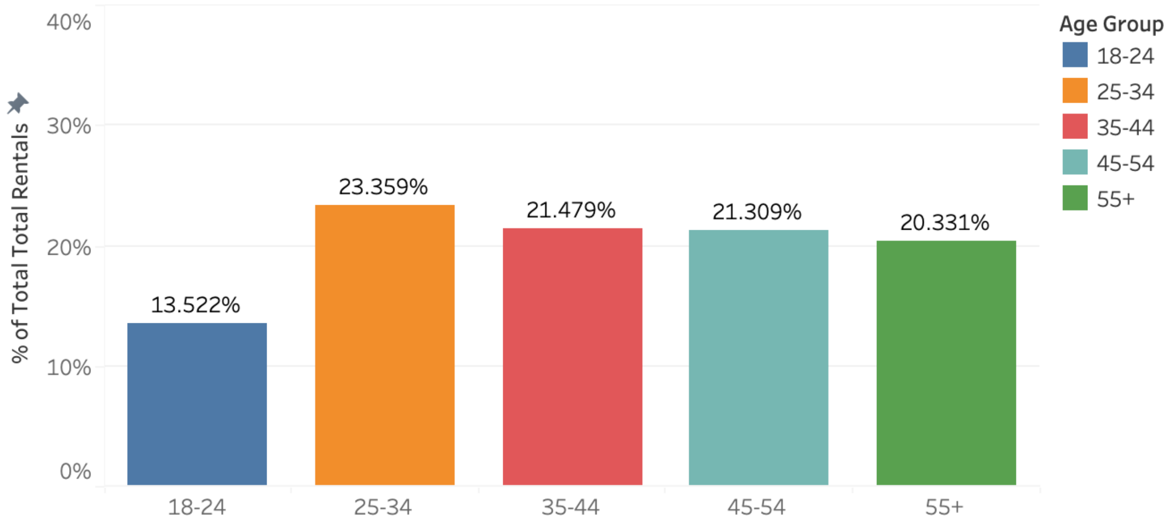


Figure 32. Percentage of Each Age Group

These reports help define our targeted customer profile as an individual likely to be aged 25–54, renting cars for commuting or business travel, and preferring affordable options. This understanding informs decisions on fleet optimisation and targeted marketing strategies.

5. Challenges

One major challenge was generating synthetic data that accurately reflected realistic business scenarios. Ensuring that service, rental, and accident repair dates did not overlap required careful coordination among the maintenance, rentals, and accidents datasets. This challenge was overcome by leveraging Python libraries such as pandas to programmatically generate and validate the data, along with custom functions to simulate vehicle usage schedules and enforce non-overlapping periods.

Additionally, some branches had missing data for 2024 due to the limited number of customers in the customers dataset. Repeat rentals were constrained so that only 30% of customers could rent more than twice. For future iterations, increasing the total number of customers or having more repeat customers is recommended to create a more comprehensive dataset and enhance reporting.

6. Conclusion

Overall, the data product successfully achieves its initial development purpose, and its robust structure allows for more identified entities and future scalability. The well-structured, normalised database enables efficient data storage and extracts vital insights, helping strategic decision-making and competitiveness in the car rental market.

7. References List

Acko, n.d. Best time to visit UK. Available at: https://www.acko.com/travel-tips/best-time-to-visit-uk/ (Accessed: 11 March 2025).

Dybka, P. (2016). Crow’s Foot Notation. [online] Vertabelo Data Modeler. Available at: https://vertabelo.com/blog/crow-s-foot-notation/ [Accessed 11 Mar. 2025].

Sikora, Z.M., 1997. Normalisation. Oracle Database Principles, pp.56-73.

The Brainy Insights (n.d.) Car Rental Market. Available at: https://www.thebrainyinsights.com/report/car-rental-market-13866#:~:text=Application%20Segment%20Analysis,off%20the%20typical%20tourist%20path (Accessed: 11 March 2025).

8. Appendix

Appendix A - Identification of Entities, Attributes, and Keys

Table A lists all identified entities and their attributes. To clearly distinguish primary and foreign keys, primary key attributes are presented in bold, while foreign key attributes are italicised.

Entities	***Attributes***
customers	**customer_id**, first_name, last_name, sex, birthdate, driver_license_id, issue_country, country_code, phone, email, street, city, county, postcode, join_date
rentals	**rental_id**, *customer_id, vehicle_id*, rental_start_date, rental_end_date, rental_duration, discount
payments	**payment_id**, *rental_id,* payment_date, payment_type, payment_amount
vehicles	**vehicle_id**, licenseplate_number, vin, *branch_id*, make, model, manufacture_year, current_mileage, daily_rate, deposit_amount, vehicle_status, next_available_date
branches	**branch_id**, branch_name, street, city, county, postcode, phone, email
insurance	**policy_number**, provider, *vehicle_id*, policy_type, premium_amount, start_date, end_date
maintenance	**maintenance_id**, *vehicle_id*, service_date, service_end_date, service_duration, service_description, service_cost, odometer_reading, service_center_id, next_maintenance_due
accidents	**accident_id**, *vehicle_id*, accident_date, accident_description, repair_start_date, repair_end_date, repair_duration, repair_cost, insurance_claim_number, claim_status
additional_services	**additional_service_id**, additional_service_name, price, pricing_unit
rental_additional_services	***rental_id, additional_service_id***, discount, final_price

Table A: A table listing identified entities and their attributes

Appendix B - Python Code

The appendix shows the completed process of creating tables, inserting synthetic data, and conducting five analyses.

Importing Libraries

```
import sqlite3
import csv
import pandas as pd
from google.colab import files
from tabulate import tabulate
```

Connect to the SQLite Database

```
# Connect to the 'fun_rides_car_rental.db' SQLite database
conn = sqlite3.connect('fun_rides_car_rental.db')
# Create a cursor for executing SQL queries
cursor = conn.cursor()
```

CREATE TABLE Queries

```
# Create the "customers" table
cursor.execute('''
CREATE TABLE IF NOT EXISTS customers (
    customer_id TEXT PRIMARY KEY NOT NULL,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    sex TEXT NOT NULL CHECK (sex IN ('M', 'F')),
    birthdate DATE NOT NULL,
    driver_license_id TEXT NOT NULL,
    issue_country TEXT NOT NULL,
    country_code TEXT NOT NULL,
    phone TEXT NOT NULL,
    email TEXT NULL,
    street TEXT NOT NULL,
    city TEXT NOT NULL,
    county TEXT NULL,
    postcode TEXT NOT NULL,
    join_date DATE NOT NULL
);
''')

# Create the "rentals" table
cursor.execute('''
CREATE TABLE IF NOT EXISTS rentals (
    rental_id INTEGER PRIMARY KEY NOT NULL,
    customer_id TEXT NOT NULL,
    vehicle_id TEXT NOT NULL,
    rental_start_date DATE NOT NULL,
    rental_end_date DATE NOT NULL,
    rental_duration INTEGER NOT NULL,
    discount REAL NOT NULL DEFAULT 0,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id)
);
''')

# Create the "payments" table
cursor.execute('''
CREATE TABLE IF NOT EXISTS payments (
    payment_id INTEGER PRIMARY KEY NOT NULL,
    rental_id INTEGER NOT NULL,
    payment_date DATE NOT NULL,
    payment_type TEXT NOT NULL CHECK (payment_type IN ('Credit Card', 'Debit Card', 'Cash', 'Bank Transfer')),
    payment_amount REAL NOT NULL,
    FOREIGN KEY (rental_id) REFERENCES rentals(rental_id)
);
''')

# Create the "vehicles" table
cursor.execute('''
CREATE TABLE IF NOT EXISTS vehicles (
    vehicle_id TEXT PRIMARY KEY NOT NULL,
    license_plate_number TEXT NOT NULL,
    vin TEXT NOT NULL,
    branch_id TEXT NOT NULL,
    make TEXT NOT NULL,
    model TEXT NOT NULL,
    manufacture_year INTEGER NOT NULL,
    current_mileage INTEGER NOT NULL,
    daily_rate REAL NOT NULL,
    deposit_amount REAL NOT NULL,
    vehicle_status TEXT NOT NULL CHECK (vehicle_status IN ('Available', 'Rented', 'Servicing', 'Totalled')),
    next_available_date DATE NULL,
    FOREIGN KEY (branch_id) REFERENCES branches(branch_id)
);
''')

# Create the "branches" table
cursor.execute('''
CREATE TABLE IF NOT EXISTS branches (
    branch_id TEXT PRIMARY KEY NOT NULL,
    branch_name TEXT NOT NULL,
    street TEXT NOT NULL,
    city TEXT NOT NULL,
    county TEXT NOT NULL,
    postcode TEXT NOT NULL,
    phone TEXT NOT NULL,
    email TEXT NOT NULL
);
''')

# Create the "insurance" table
cursor.execute('''
CREATE TABLE IF NOT EXISTS insurance (
    policy_number TEXT PRIMARY KEY NOT NULL,
    provider TEXT NOT NULL,
    vehicle_id TEXT NOT NULL,
    policy_type TEXT NOT NULL,
    premium_amount REAL NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id)
);
''')

# Create the "maintenance" table
cursor.execute('''
CREATE TABLE IF NOT EXISTS maintenance (
    maintenance_id TEXT PRIMARY KEY NOT NULL,
    vehicle_id TEXT NOT NULL,
    service_date DATE NOT NULL,
    service_end_date DATE NULL,
    service_duration INTEGER NULL,
    service_description TEXT NULL,
    service_cost REAL NULL,
    odometer_reading INTEGER NOT NULL,
    service_center_id TEXT NOT NULL,
    next_maintenance_due INTEGER NOT NULL,
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id)
);
''')

# Create the "accidents" table
cursor.execute('''
CREATE TABLE IF NOT EXISTS accidents (
    accident_id TEXT PRIMARY KEY NOT NULL,
    vehicle_id TEXT NOT NULL,
    accident_date DATE NOT NULL,
    accident_description TEXT NOT NULL,
    repair_start_date DATE NULL,
    repair_end_date DATE NULL,
    repair_duration INTEGER NULL,
    repair_cost REAL NULL,
    insurance_claim_number TEXT NULL,
    claim_status TEXT NULL,
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id)
);
''')

# Create the "additional_services" table
cursor.execute('''
CREATE TABLE IF NOT EXISTS additional_services (
    additional_service_id INTEGER PRIMARY KEY NOT NULL,
    additional_service_name TEXT NOT NULL,
    price REAL NULL,
    pricing_unit TEXT NULL
);
''')

# Create the "rental_additional_services" table (junction table)
cursor.execute('''
CREATE TABLE IF NOT EXISTS rental_additional_services (
    rental_id INTEGER NOT NULL,
    additional_service_id INTEGER NOT NULL,
    discount REAL NOT NULL DEFAULT 0
    final_price REAL NOT NULL,
    PRIMARY KEY (rental_id, additional_service_id),
    FOREIGN KEY (rental_id) REFERENCES rentals(rental_id),
    FOREIGN KEY (additional_service_id) REFERENCES additional_services(additional_service_id)
);
''')

conn.commit()
print("Database and tables created successfully.")
```

```
Database and tables created successfully.
```

Database Schema *Check*

(Outputs are Presented in Page 4 - 8)

```
# Get table names from the schema
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()

for table_name in tables:
    table = table_name[0]
    print(f"Table: {table}")

    # Get column info for the current table
    cursor.execute(f"PRAGMA table_info({table});")
    columns = cursor.fetchall()

    # Get foreign key info for the current table
    cursor.execute(f"PRAGMA foreign_key_list({table});")
    foreign_keys = cursor.fetchall()

    # foreign_keys tuple structure: (id, seq, ref_table, fk_from, ref_column, on_update, on_delete, match)
    # Build a dictionary mapping each foreign key column to its referenced table(s)
    fk_ref = {}
    for fk in foreign_keys:
        col_from = fk[3]      # the "from" column in the current table
        ref_table = fk[2]     # the referenced table name
        if col_from in fk_ref:
            fk_ref[col_from].append(ref_table)
        else:
            fk_ref[col_from] = [ref_table]

    # Define headers for tabulate output
    headers = ["Column", "Type", "Not Null", "Default Value", "Primary Key", "Foreign Key", "Reference Table"]
    rows = []
    for col in columns:
        # col structure: (cid, name, type, notnull, dflt_value, pk)
        column_name = col[1]
        is_fk = 1 if column_name in fk_ref else 0
        ref_tables = ", ".join(fk_ref[column_name]) if column_name in fk_ref else ""
        rows.append([column_name, col[2], col[3], col[4], col[5], is_fk, ref_tables])

    # Print the table information using tabulate
    print(tabulate(rows, headers=headers, tablefmt="pretty"))
    print("-" * 40)
```

Show hidden output

Load the Synthetic Data

```
# Upload the synthetic data
uploaded = files.upload()
```

Show hidden output

```
def import_csv_to_table(csv_file, table_name):
    # Open the CSV file and create a reader, skipping the header row
    with open(csv_file, 'r', encoding='utf-8') as file:
        csv_reader = csv.reader(file)
        next(csv_reader)  # Skip header row if present
        # For each row in the CSV, build and execute an INSERT query
        for row in csv_reader:
            placeholders = ', '.join(['?' for _ in row])
            sql = f"INSERT INTO {table_name} VALUES ({placeholders})"
            cursor.execute(sql, row)

try:
    import_csv_to_table('branches.csv', 'branches')
    import_csv_to_table('vehicles.csv', 'vehicles')
    import_csv_to_table('customers.csv', 'customers')
    import_csv_to_table('additional_services.csv', 'additional_services')
    import_csv_to_table('rentals.csv', 'rentals')
    import_csv_to_table('payments.csv', 'payments')
    import_csv_to_table('accidents.csv', 'accidents')
    import_csv_to_table('maintenance.csv', 'maintenance')
    import_csv_to_table('insurance.csv', 'insurance')
    import_csv_to_table('rental_additional_services.csv', 'rental_additional_services')

    conn.commit()
    print("Data imported successfully!")
except Exception as e:
    print(f"An error occurred: {e}")
    conn.rollback()
```

```
Data imported successfully!
```

Data Integrity Checks

Record Count Check

```
csv_files = {
    "customers": "customers.csv",
    "rentals": "rentals.csv",
    "payments": "payments.csv",
    "vehicles": "vehicles.csv",
    "branches": "branches.csv",
    "insurance": "insurance.csv",
    "maintenance": "maintenance.csv",
    "accidents": "accidents.csv",
    "additional_services": "additional_services.csv",
    "rental_additional_services": "rental_additional_services.csv"
}


# Dictionary to store record counts
csv_counts = {}

# Count the number of records in each CSV file
for table, file_path in csv_files.items():
    try:
        df = pd.read_csv(file_path)
        csv_counts[table] = len(df)
    except Exception as e:
        csv_counts[table] = f"Error: {e}"


# List of tables to check for record counts
tables_to_check = [
    'customers',
    'rentals',
    'payments',
    'vehicles',
    'branches',
    'insurance',
    'maintenance',
    'accidents',
    'additional_services',
    'rental_additional_services'
]

# Dictionary to store database record counts
db_counts = {}


# Iterate through each table in the list
for table in tables_to_check:
    try:
        # Execute a SQL query to count the records in the current table
        df_count = pd.read_sql_query(f"SELECT COUNT(*) AS total_records FROM {table};", conn)
        db_counts[table] = df_count['total_records'].iloc[0]
    except Exception as e:
        db_counts[table] = f"Error: {e}"

# Create a DataFrame for comparison
comparison_df = pd.DataFrame({
    "Table": tables_to_check,
    "CSV Record Count": [csv_counts.get(table, "N/A") for table in tables_to_check],
    "DB Record Count": [db_counts.get(table, "N/A") for table in tables_to_check]
})

print(comparison_df)
```

```
                        Table  CSV Record Count  DB Record Count
0                   customers             22498            22498
```

```
1                      rentals         27539          27539
2                     payments         33182          33182
3                     vehicles           150            150
4                     branches            10             10
5                    insurance           587            587
6                  maintenance           726            726
7                    accidents            32             32
8          additional_services             7              7
9  rental_additional_services         80068          80068
```

Unique Primary Key Check

```
# Map each table to its primary key column(s). For composite keys, columns are concatenated
tables_and_keys = {
    'accidents': 'accident_id',
    'additional_services': 'additional_service_id',
    'branches': 'branch_id',
    'customers': 'customer_id',
    'insurance': 'policy_number',
    'maintenance': 'maintenance_id',
    'payments': 'payment_id',
    'rental_additional_services': 'rental_id || additional_service_id',  # Composite key
    'rentals': 'rental_id',
    'vehicles': 'vehicle_id'
}

# Loop through each table, grouping by the primary key and checking for duplicates
for table, key in tables_and_keys.items():
    query = f"""
    SELECT {key} AS key_val, COUNT(*) AS count
    FROM {table}
    GROUP BY key_val
    HAVING COUNT(*) > 1;
    """

    # Execute the query and fetch results
    cursor.execute(query)
    duplicates = cursor.fetchall()

    # Output the results
    if duplicates:
        print(f"Duplicates found in table '{table}':")
        for row in duplicates:
            print(row)
    else:
        print(f"No duplicates in table '{table}'.")
```

```
No duplicates in table 'accidents'.
No duplicates in table 'additional_services'.
No duplicates in table 'branches'.
No duplicates in table 'customers'.
No duplicates in table 'insurance'.
No duplicates in table 'maintenance'.
No duplicates in table 'payments'.
No duplicates in table 'rental_additional_services'.
No duplicates in table 'rentals'.
No duplicates in table 'vehicles'.
```

Invalid Value Check

```
# Check for invalid sex values in customers
cursor.execute("SELECT customer_id, sex FROM customers WHERE sex NOT IN ('M', 'F');")
invalid_sex = cursor.fetchall()
if invalid_sex:
    print("Invalid sex values found in customers:")
    for row in invalid_sex:
        print(row)
else:
    print("All 'sex' values in customers are valid.")

# Check for invalid vehicle_status in vehicles
cursor.execute("""
    SELECT vehicle_id, vehicle_status
    FROM vehicles
    WHERE vehicle_status NOT IN ('Available', 'Rented', 'Servicing', 'Totalled');
""")
invalid_status = cursor.fetchall()
if invalid_status:
    print("Invalid vehicle_status values found in vehicles:")
    for row in invalid_status:
        print(row)
else:
    print("All 'vehicle_status' values in vehicles are valid.")

# Check for invalid payment_type in payments
```

```
cursor.execute("""
    SELECT payment_id, payment_type
    FROM payments
    WHERE payment_type NOT IN ('Credit Card', 'Debit Card', 'Bank Transfer', 'Cash');
""")
invalid_payment = cursor.fetchall()
if invalid_payment:
    print("Invalid payment_type values found in payments:")
    for row in invalid_payment:
        print(row)
else:
    print("All 'payment_type' values in payments are valid.")
```

```
All 'sex' values in customers are valid.
All 'vehicle_status' values in vehicles are valid.
All 'payment_type' values in payments are valid.
```

Rentals Integrity Check

SQL query to find overlapping rentals for the same vehicle using date comparisons

```
query = """
SELECT
  r1.rental_id AS rental1,
  r2.rental_id AS rental2,
  r1.vehicle_id,
  date(r1.rental_start_date) AS start1, date(r1.rental_end_date) AS end1,
  date(r2.rental_start_date) AS start2, date(r2.rental_end_date) AS end2
FROM rentals r1
JOIN rentals r2
  ON r1.vehicle_id = r2.vehicle_id
 AND r1.rental_id < r2.rental_id
WHERE date(r1.rental_start_date) <= date(r2.rental_end_date)
  AND date(r2.rental_start_date) <= date(r1.rental_end_date);
"""

# Execute the query and fetch results
cursor.execute(query)
overlaps = cursor.fetchall()

# Output the results
if overlaps:
    print("Overlapping rentals for the same vehicle:")
    for overlap in overlaps:
        print(overlap)
else:
    print("No overlapping rentals for the same vehicle found.")
```

```
No overlapping rentals for the same vehicle found.
```

SQL query to find overlapping rentals for the same customer using date comparisons

```
query = """
SELECT
  r1.rental_id AS rental1,
  r2.rental_id AS rental2,
  r1.customer_id,
  date(r1.rental_start_date) AS start1, date(r1.rental_end_date) AS end1,
  date(r2.rental_start_date) AS start2, date(r2.rental_end_date) AS end2
FROM rentals r1
JOIN rentals r2
  ON r1.customer_id = r2.customer_id
 AND r1.rental_id < r2.rental_id
WHERE date(r1.rental_start_date) <= date(r2.rental_end_date)
  AND date(r2.rental_start_date) <= date(r1.rental_end_date);
"""

# Execute the query and fetch results
cursor.execute(query)
overlapping_customer_rentals = cursor.fetchall()

# Output the results
if overlapping_customer_rentals:
    print("Overlapping rentals for the same customer:")
    for overlap in overlapping_customer_rentals:
        print(overlap)
else:
    print("No overlapping rentals for the same customer found.")
```

```
No overlapping rentals for the same customer found.
```

Business Reports

These queries have output several tables and these tables are presented in the Business Report section.

Financial Analysis

1. Branches Revenue Report 2024

```
# Define the query
query = """
SELECT
    b.branch_id,
    b.branch_name,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '01' THEN p.payment_amount ELSE 0 END) AS Jan_2024,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '02' THEN p.payment_amount ELSE 0 END) AS Feb_2024,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '03' THEN p.payment_amount ELSE 0 END) AS Mar_2024,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '04' THEN p.payment_amount ELSE 0 END) AS Apr_2024,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '05' THEN p.payment_amount ELSE 0 END) AS May_2024,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '06' THEN p.payment_amount ELSE 0 END) AS Jun_2024,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '07' THEN p.payment_amount ELSE 0 END) AS Jul_2024,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '08' THEN p.payment_amount ELSE 0 END) AS Aug_2024,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '09' THEN p.payment_amount ELSE 0 END) AS Sep_2024,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '10' THEN p.payment_amount ELSE 0 END) AS Oct_2024,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '11' THEN p.payment_amount ELSE 0 END) AS Nov_2024,
    SUM(CASE WHEN strftime('%m',
         substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2)
         ) = '12' THEN p.payment_amount ELSE 0 END) AS Dec_2024,
    SUM(p.payment_amount) AS Total_Annual_Revenue
FROM branches b
JOIN vehicles v ON b.branch_id = v.branch_id
JOIN rentals r ON v.vehicle_id = r.vehicle_id
JOIN payments p ON r.rental_id = p.rental_id
WHERE (substr(p.payment_date, 7, 4) || '-' || substr(p.payment_date, 4, 2) || '-' || substr(p.payment_date, 1, 2))
      BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY
    b.branch_id,
    b.branch_name
ORDER BY
    b.branch_id;
"""

# Execute the query and load the results into a DataFrame
df_revenue = pd.read_sql_query(query, conn)

# Print the report
print("Annual Revenue Report for 2024 by Branch:")
print(df_revenue)
```

Show hidden output

2. Year-Over-Year (YoY) and Quarter-over-Quarter (QoQ) Revenue Growth Report

```
# Define the query
query = """
WITH quarterly_revenue AS (
    SELECT
        CAST(substr(r.rental_start_date, 7, 4) AS INTEGER) AS rental_year,
        CASE
            WHEN CAST(substr(r.rental_start_date, 4, 2) AS INTEGER) BETWEEN 1 AND 3 THEN 'Q1'
            WHEN CAST(substr(r.rental_start_date, 4, 2) AS INTEGER) BETWEEN 4 AND 6 THEN 'Q2'
            WHEN CAST(substr(r.rental_start_date, 4, 2) AS INTEGER) BETWEEN 7 AND 9 THEN 'Q3'
            ELSE 'Q4'
        END AS rental_quarter,
        SUM(p.payment_amount) AS total_revenue
    FROM rentals r
    JOIN payments p ON r.rental_id = p.rental_id
    JOIN rental_additional_services ras ON r.rental_id = ras.rental_id
    JOIN additional_services as2 ON ras.additional_service_id = as2.additional_service_id
    GROUP BY rental_year, rental_quarter
)

SELECT
    curr.rental_year || '-' || curr.rental_quarter AS rental_period,
    curr.total_revenue,

    -- YoY Calculation
    prev_year.total_revenue AS prev_year_revenue,
    CASE
        WHEN prev_year.total_revenue IS NOT NULL
        THEN ROUND(((curr.total_revenue - prev_year.total_revenue) / prev_year.total_revenue) * 100, 2)
        ELSE NULL
    END AS yoy_growth_percentage,

    -- QoQ Calculation
    prev_quarter.total_revenue AS prev_quarter_revenue,
    CASE
        WHEN prev_quarter.total_revenue IS NOT NULL
        THEN ROUND(((curr.total_revenue - prev_quarter.total_revenue) / prev_quarter.total_revenue) * 100, 2)
        ELSE NULL
    END AS qoq_growth_percentage

FROM quarterly_revenue curr

-- Join for YoY growth (previous year, same quarter)
LEFT JOIN quarterly_revenue prev_year
ON curr.rental_quarter = prev_year.rental_quarter
AND curr.rental_year = prev_year.rental_year + 1

-- Join for QoQ growth (previous quarter, same year or previous year if crossing Q4 to Q1)
LEFT JOIN quarterly_revenue prev_quarter
ON (
    (curr.rental_year = prev_quarter.rental_year AND
        ((curr.rental_quarter = 'Q2' AND prev_quarter.rental_quarter = 'Q1') OR
         (curr.rental_quarter = 'Q3' AND prev_quarter.rental_quarter = 'Q2') OR
         (curr.rental_quarter = 'Q4' AND prev_quarter.rental_quarter = 'Q3')))
    OR
    (curr.rental_quarter = 'Q1' AND prev_quarter.rental_quarter = 'Q4' AND curr.rental_year = prev_quarter.rental_year + 1)
)

ORDER BY curr.rental_year, curr.rental_quarter;
"""

# Execute the query and load the results into a DataFrame
df_annual_rev = pd.read_sql_query(query, conn)

# Print the report
print("Annual Revenue Report:")
print(df_annual_rev)
```

Show hidden output

Vehicle Utilisation Report

```
# Define the query
query = """
SELECT
    b.branch_id,
    b.branch_name,
    AVG(
      CASE
         WHEN (366 - COALESCE(m.total_maint, 0) - COALESCE(a.total_acc, 0)) > 0
         THEN COALESCE(r.total_rented, 0) * 100.0 / (366 - COALESCE(m.total_maint, 0) - COALESCE(a.total_acc, 0))
         ELSE 0
      END
    ) AS vehicle_utilization
FROM vehicles v
JOIN branches b ON v.branch_id = b.branch_id
LEFT JOIN (
  SELECT
    vehicle_id,
    SUM(
      CASE
         WHEN (substr(rental_end_date, 7, 4) || '-' || substr(rental_end_date, 4, 2) || '-' || substr(rental_end_date, 1, 2)) > '2024-12-31'
         THEN julianday('2024-12-31') - julianday(substr(rental_start_date, 7, 4) || '-' || substr(rental_start_date, 4, 2) || '-' || substr(rental_st
         ELSE rental_duration
      END
    ) AS total_rented
  FROM rentals
  WHERE (substr(rental_start_date, 7, 4) || '-' || substr(rental_start_date, 4, 2) || '-' || substr(rental_start_date, 1, 2))
         BETWEEN '2024-01-01' AND '2024-12-31'
  GROUP BY vehicle_id
) r ON v.vehicle_id = r.vehicle_id
LEFT JOIN (
  SELECT vehicle_id, SUM(service_duration) AS total_maint
  FROM maintenance
  WHERE (substr(service_date, 7, 4) || '-' || substr(service_date, 4, 2) || '-' || substr(service_date, 1, 2))
         BETWEEN '2024-01-01' AND '2024-12-31'
  GROUP BY vehicle_id
) m ON v.vehicle_id = m.vehicle_id
LEFT JOIN (
  SELECT vehicle_id, SUM(repair_duration) AS total_acc
  FROM accidents
  WHERE (substr(accident_date, 7, 4) || '-' || substr(accident_date, 4, 2) || '-' || substr(accident_date, 1, 2))
         BETWEEN '2024-01-01' AND '2024-12-31'
  GROUP BY vehicle_id
) a ON v.vehicle_id = a.vehicle_id
GROUP BY b.branch_id, b.branch_name
ORDER BY b.branch_id;
"""

# Execute the query and load the results into a DataFrame
df_utilization = pd.read_sql_query(query, conn)

# Print the report
print("Vehicle Utilization Report for 2024):")
print(df_utilization)
```

Show hidden output

Demand Analysis

1. Customer Demographics

```
# Define the query
query = """
SELECT
    CASE
        WHEN (2024 - CAST(substr(birthdate, 7, 4) AS INTEGER)) BETWEEN 18 AND 24 THEN '18-24'
        WHEN (2024 - CAST(substr(birthdate, 7, 4) AS INTEGER)) BETWEEN 25 AND 34 THEN '25-34'
        WHEN (2024 - CAST(substr(birthdate, 7, 4) AS INTEGER)) BETWEEN 35 AND 44 THEN '35-44'
        WHEN (2024 - CAST(substr(birthdate, 7, 4) AS INTEGER)) BETWEEN 45 AND 54 THEN '45-54'
        ELSE '55+'
    END AS age_group,
    COUNT(*) AS total_customers
FROM customers
GROUP BY age_group
ORDER BY age_group;
"""

# Execute the query and load the results into a DataFrame
df_age_group = pd.read_sql_query(query, conn)

#Print the report
print("Rental counts by age group for each vehicle:")
print(df_age_group)
```

Show hidden output

2. Vehicle Demand by Gender

```
# Define the query
query = """
WITH rental_count AS (
    SELECT
        r.vehicle_id,
        c.sex,
        COUNT(r.rental_id) AS total_rentals
    FROM rentals r
    JOIN customers c ON r.customer_id = c.customer_id
    GROUP BY r.vehicle_id, c.sex
)
SELECT
    v.make,
    v.model,
    rc.sex,
    rc.total_rentals
FROM rental_count rc
JOIN vehicles v ON rc.vehicle_id = v.vehicle_id
```

```
GROUP BY v.make, v.model, rc.sex
ORDER BY v.make, v.model, rc.sex;
"""

# Execute the query and load the results into a DataFrame
df_rental_by_sex = pd.read_sql_query(query, conn)

# Print the report
print("Rental Counts by Vehicle and Customer Sex:")
print(df_rental_by_sex)
```

Show hidden output

3. Average Rental Frequency

```
# Define the query
query = """
WITH rental_count AS (
    SELECT
        vehicle_id,
        COUNT(*) AS rental_times
    FROM rentals
    WHERE rental_start_date >= '2024-01-01' AND rental_start_date <= '2024-12-31'
    GROUP BY vehicle_id
)
SELECT
    v.make,
    v.model,
    SUM(r.rental_times) AS total_rentals,
    COUNT(r.vehicle_id) AS total_vehicles,
    AVG(r.rental_times) AS avg_rental_frequency
FROM rental_count r
JOIN vehicles v ON r.vehicle_id = v.vehicle_id
GROUP BY v.make, v.model
ORDER BY avg_rental_frequency DESC;
"""

# Execute the query and load the results into a DataFrame
df_avg_rental = pd.read_sql_query(query, conn)

# Print the report
print("Average Rental Frequency Report:")
print(df_avg_rental)
```

Show hidden output

```
# Close the connection
conn.close()
```